Pergamon

Computers Math. Applic. Vol. 31, No. 6, pp. 97–115, 1996

0898-1221/96 $15.00 + 0.00

S0898-1221(96)00009-0

A Comparison of Some Adaptive Space Mesh Solvers for the Numerical Solution of Parabolic Partial Differential Equations

J. CARROLL
Dublin City University, Dublin 9, Ireland
CarrollJ@DCU.IE

S. STEWART
Mathematical Sciences, Dublin City University
Dublin 9, Ireland

(Received May 1995; accepted June 1995)

Abstract—We examine a number of adaptive space mesh routines which were designed to provide accurate numerical approximations to the solutions of parabolic partial differential equations in one space dimension. Both static and dynamic regridding techniques are considered, and in numerical experiments, we compare their relative performances by measuring a range of statistics including accuracy and cpu when applied to approximate the solutions of a set of representative test problems.

Keywords—Partial differential equations, Time-dependent problems, Method of lines, Moving grids, Adaptive solvers.

1. INTRODUCTION

This paper is concerned with the comparison of a number of efficient numerical solution procedures which were designed for a broad class of initial value problems coming from a variety of application areas including the transient simulation of silicon devices and circuits, diffusion, combustion modelling, the simulation of gas-transmission networks, interstellar ionization fronts, and supernova blast waves. Such problems can be modelled mathematically as systems of partial differential equations (PDEs) in one space variable and time which describe the evolution from a given initial configuration. The PDEs are usually of parabolic type, but in some cases are of hyperbolic type which can be treated numerically as parabolic by adding 'artificial viscosity' terms.

In the conventional approach to the numerical solution of such problems, the system of PDEs is discretized on a fixed grid or mesh of points in space, and the solution, as represented on this grid, is then advanced in time either by an equivalent time discretization or by regarding the spatially discretized system as a large coupled system of ordinary differential equations (the method of lines).

The great disadvantage of this approach, for the class of problems in which we are interested, is that to adequately resolve the steep localized gradients which can occur in the solutions, the grid has to be very fine. However, if the grid is uniformly spaced, this means that a very large number

We are indebted to J. E. Flaherty (Rensselaer Polytechnic Institute), R. D. Russell (Simon Fraser University), W. E. Schiesser (Lehigh University), and J. Verwer (CWI, Amsterdam) for providing us with their codes and for the advice and many interesting communications which we shared during the course of our experiments.

of grid points are required and that most of these are located in regions where the solution is quite smooth.

The obvious solution is to use a nonuniform distribution of grid points so that high spatial resolution is provided only in those regions where the nonlinear nature of the problem requires it. This is the fundamental concept of adaptive meshing. Similar techniques have been widely used to solve problems involving ordinary differential equations with rapid transitions, and are thus likely candidates for providing the computational methods and codes necessary to solve more difficult problems involving partial differential equations.

One can distinguish two approaches to adaptive meshing:

(1) Local refinement methods: fine grids are added to coarse regions where the solution is not adequately resolved.
(2) Moving mesh methods: grids of a fixed number of finite difference cells or finite elements are moved so as to follow local nonuniformities in the solution.

While both methods have their advantages, the local refinement technique is intrinsically linked to the concept of multigrid methods and leads to considerable technical complications in implementation. The codes we consider use static regridding, finite difference, and finite element discretizations for the Lagrangian form of the problem, and a moving finite element method.

In the next section, we provide a brief description of the codes, while in Section 3, we describe the results of a number of experiments, listing test problems and tables of results. Some general observations on the relative performances are summarised in Section 4.

2. BRIEF DESCRIPTION OF THE CODES

2.1. The CWI Code

The CWI code contains an implementation of a finite-element discretization for the Lagrangian form of the general parabolic partial differential equation which was written by Verwer and his research group at CWI (see [1], for example). The grid equations and the discrete differential equations are solved simultaneously by Petzold's differential-algebraic system solver DASSL [2].

The CWI code was designed to solve time-dependent partial differential equations in one space dimension, having the general form

$$\sum_{k=1}^{\mathrm{NPDE}} C_{j,k}\left(x,t,u,u_x\right)\frac{\partial u^k}{\partial t} = x^{-m}\frac{\partial}{\partial x}\left(x^m R_j\left(x,t,u,u_x\right)\right) - Q_j\left(x,t,u,u_x\right), \tag{1}$$

for $j = 1,\ldots,\mathrm{NPDE}$ and $x \in [x_L, x_R]$, $t > t_0$ and $m \in \{0,1,2\}$, where NPDE denotes the number of PDEs, u is the solution vector and R_j and Q_j can be thought of, in special cases, as flux and source terms, respectively. The user specifies boundary conditions in the form

$$\beta_j(x,t)R_j\left(x,t,u,u_x\right) = \gamma_j\left(x,t,u,u_x\right)$$

at $x = x_L$ and $x = x_R$ for $j = 1,\ldots,\mathrm{NPDE}$. The initial conditions must satisfy

$$u\left(x,t_0\right) = u^0(x), \qquad x \in [x_L, x_R].$$

For a numerical solution, the partial differential equation

$$u_t = f(u,x,t), \qquad x_L < x < x_R, \quad t > 0, \tag{2}$$

is first transformed into its Lagrangian form

$$\dot{u} - u_x\dot{x} = f(u,x,t), \tag{3}$$

where $\dot{u}$ denotes the total time derivative $\dot{u} = \frac{du}{dt} = u_t + u_x\dot{x}$. Then, N time dependent grid points

$$x_L = X_0 < \cdots < X_i(t) < X_{i+1}(t) < \cdots < X_{N+1} = x_R \tag{4}$$

are selected and equation (3) is discretized in space to obtain

$$\dot{U}_i - \frac{U_{i+1} - U_{i-1}}{X_{i+1} - X_{i-1}}\dot{X}_i = F_i, \qquad t > t_0, \quad 1 \le i \le N. \tag{5}$$

Here, U_i and F_i represent the semidiscrete approximation to the exact PDE solution u and the right-hand side function $f(u, x, t)$, respectively, at the point $(x, t) = (X_i(t), t)$. To solve the ODE system (5), additional equations are required for the time dependent grid points X_i which are as yet unknown (see [1,3] for details, and the next subsection for some examples) and the combined systems may be written in the form

$$\begin{aligned} \dot{U} - D\dot{X} &= F, \\ \tau B\dot{X} &= g, \end{aligned} \tag{6}$$

where D and B are solution dependent matrices, and F and g are solution dependent vectors. The system (6) can then be rearranged in the linearly implicit form

$$A(Y)\dot{Y} = L(Y),$$

where

$$Y := \left(\ldots, U_i^1, \ldots, U_i^{\mathrm{NPDE}}, \ldots, X_i, \ldots\right)^{\mathsf{T}}.$$

The CWI package employs a second-order, nonlinear Galerkin spatial discretization method, and the approach is justified by the authors (see [1,3]) because of the potential need to reduce accuracy problems that arise for coefficients like x^{-m} in (1). The discretization method is reported extensively in [4], and for an account of how the discretization is applied to the PDE class when transformed to its Lagrangian form, we refer the interested reader to [3].

2.2. The SFU Code

The SFU code was developed by Huang, Ren, and Russell [5] to develop and test several moving mesh partial differential equations (MMPDEs) based on the equidistribution principle. Although some of the moving mesh methods they consider are related to previously studied techniques, there are key differences, and in addition, a novel approach to spatial smoothing is also considered (we refer the interested reader to [5] for further details).

As with the development of the CWI method given earlier, the partial differential equation

$$u_t = f(u, x, t), \qquad x_L < x < x_R, \quad t > 0,$$

is transformed into its Lagrangian form

$$\dot{u} - u_x\dot{x} = f(u, x, t),$$

where $\dot{u}$ denotes the total time derivative $\dot{u} = \frac{du}{dt} = u_t + u_x\dot{x}$. The N time dependent grid points are defined as

$$x_L = X_0 < \cdots < X_i(t) < X_{i+1}(t) < \cdots < X_{N+1} = x_R, \tag{7}$$

and a spatial discretization yields

$$\dot{U}_i - \frac{U_{i+1} - U_{i-1}}{X_{i+1} - X_{i-1}}\dot{X}_i = F_i, \qquad t > t_0, \quad 1 \le i \le N. \tag{8}$$

Here, U_i and F_i represent the semidiscrete approximation to the exact PDE solution u and the right-hand side function $f(u, x, t)$, respectively, at the point $(x, t) = (X_i(t), t)$. As before, to solve

this ODE system, additional equations are required for the time dependent grid points X_i which are as yet unknown.

With x and ξ denoting the physical and computational coordinates, respectively, assumed without loss of generality to be over the unit interval $[0,1]$, and $M(x,t)$ a monitor function which provides some measure of the computational error in the solution of the underlying physical PDE, Huang, Ren, and Russell [5] present a variety of MMPDEs, namely

$$\frac{\partial^2}{\partial \xi^2}(M\dot{x}) = -\frac{1}{\tau}\frac{\partial}{\partial \xi}\left(M\frac{\partial x}{\partial \xi}\right), \tag{9}$$

$$\frac{\partial}{\partial \xi}\left(M\frac{\partial \dot{x}}{\partial \xi}\right) = -\frac{1}{\tau}\frac{\partial}{\partial \xi}\left(M\frac{\partial x}{\partial \xi}\right), \tag{10}$$

$$-\dot{x} = -\frac{1}{\tau}\frac{\partial}{\partial \xi}\left(M\frac{\partial x}{\partial \xi}\right), \tag{11}$$

$$\frac{\partial^2 \dot{x}}{\partial \xi^2} = -\frac{1}{\tau}\frac{\partial}{\partial \xi}\left(M\frac{\partial x}{\partial \xi}\right), \tag{12}$$

$$\frac{\partial}{\partial \xi}\left(M\frac{\partial \dot{x}}{\partial \xi}\right) - 2\frac{\partial}{\partial \xi}\left(M\frac{\partial x}{\partial \xi}\right)\frac{\frac{\partial \dot{x}}{\partial \xi}}{\frac{\partial x}{\partial \xi}} = -\frac{1}{\tau}\frac{\partial}{\partial \xi}\left(M\frac{\partial x}{\partial \xi}\right), \tag{13}$$

which not only force the mesh $(x(\xi,t))$ toward equidistribution, but also prevent the mesh from crossing. They are discretized in space with centered finite differences on the uniform (in ξ) computational mesh and are solved, coupled with the ODE system (8), in the usual method of lines approach using Petzold's differential-algebraic system solver DASSL [2].

2.3. The RPI Code

The RPI code was written by Adjerid and Flaherty [6] at the Rensselaer Polytechnic Institute to approximate the solutions of systems of parabolic partial differential equations using a moving Galerkin finite element method for the spatial discretization, and DASSL [2] for the temporal integration. The package solves problems of the form

$$Lu := M(x,t)u_t + f(x,t,u,u_x) - [D(x,t,u)u_x]_x = 0, \qquad 0 < x < 1, \quad t > 0, \tag{14}$$

subject to the initial condition

$$u(x,0) = u^0(x), \qquad 0 \le x \le 1, \tag{15}$$

and boundary conditions of the form

$$\text{either } u_i(x,t) = a_i(t) \quad \text{or} \quad \sum_{j=1}^{m} D_{ij}u_{ji}(x,t) = a_i(t), \tag{16}$$

at $x = 0$ and $x = 1$, for $t > 0$ and $i = 1, 2, \ldots, m$. Here, D and M are positive definite.

The user has to provide procedures for evaluating $f(x,t,u,u_x)$, $M(x,t)$, $D(x,t)$, and the initial and boundary conditions. A temporal error tolerance and parameters required by DASSL are internally provided by the code.

The equations (14)–(16) are discretized in space using the finite element Galerkin procedure with piecewise linear approximations on a moving mesh. At the same time, an error estimate is computed using a piecewise quadratic correction. This error estimate is used to move the mesh so that it approximately equidistributes the local spatial component of the discretization error (in H^1).

In [6], the authors stress that their procedure differs from the moving finite element method of Miller and Miller [7] in that they move the mesh so that the spatial error in H^1 is equidistributed

rather than moving the mesh so as to minimize the residual in L_2. The mesh equidistributes the local error, but mesh refinement, when it becomes necessary, is performed globally. Adjerid and Flaherty [6] point out that this approach does not require the use of complicated tree data structures which are prevalent in many other local refinement schemes.

By constructing a weak form of (14)–(16), namely

$$(v, Mu_t) + (v, f) + a(v, u) = 0, \tag{17}$$

for all $v \in H_0^1$, where, as usual,

$$(v, u) = \int_0^1 v(x,t)^\top u(x,t)\, dx,$$
$$a(v, u) = \int_0^1 v_x^\top D(x,t,u)u_x\, dx - v^\top D(x,t,u)u_x\big|_0^1,$$

the method necessitates the solution of three sets of ordinary differential equations to obtain the finite element solutions of (17). With U and V denoting the finite-dimensional approximations to u and v, respectively, they are as follows. The first set for the finite element solution U is

$$\begin{aligned}(V, MU_t) + (V, f) + a(V, U) &= 0,\\ (V, U) &= \left(V, u^0\right).\end{aligned} \tag{18}$$

The second set of equations is for the quadratic error estimate, i.e., the finite dimensional approximation E to the the error in the piecewise linear finite element equations, $e(x,t) = u(x,t) - U(x,t)$. It is given as follows:

$$\begin{aligned}(V, M(U_t + E_t)) + (V, f(t, U + E, U_x + E_x)) + a(V, U + E) &= 0,\\ (V, E) &= \left(V, u^0 - U\right).\end{aligned} \tag{19}$$

The third set to determine the corresponding mesh points is

$$\dot{x}_{i+1} - 2\dot{x}_i + \dot{x}_{i+1} = -\lambda\left(\|E_{i+1}\Psi_{i+1}\| - \|E_i\Psi_i\|\right), \tag{20}$$

where λ is a positive constant and Ψ is a piecewise quadratic basis function.

As stated earlier, the temporal integration for the three sets of ODEs (18)–(20) is performed using DASSL [2] and it is halted at specified times by the code when an error estimate is examined. If it is larger than a specified tolerance, the step is rejected and the integration is redone using a finer spatial discretization. On the other hand, if the error estimate indicates that the solution is being calculated too accurately, then the integration is continued with a coarser spatial mesh.

2.4. DSS/2

The package DSS/2 (called LSODE2 in [8]) provides an adaptive grid solution of a one-dimensional parabolic partial differential equation and the time integration is achieved using LSODE [9,10].

DSS/2 uses cubic splines in space to create a semidiscretized system which can be integrated by LSODE. Using the numerical method of lines, the code has a facility to allow the number of ODEs defined by the adaptive grid to change as the grid evolves.

The sample program in [8] for the numerical solution of Burgers' equation calls a set of user-supplied subroutines, namely INITAL, DERV and PRINT, plus DATA (which provides such information as the start and final times, the print interval and the number of mesh points). The initial conditions are set in subroutine INITAL while the derivatives are programmed in subroutine DERV.

The code is organized so that each call to subroutine LSODE covers one print interval. The numerical solution is printed and plotted in subroutine PRINT and this subroutine also calls ANUGB1 to redefine the adaptive grid at each print time. ANUGB1 in turn calls a series of subordinate routines provided with DSS/2. This redefinition of the adaptive grid in effect also defines a new ODE problem and so LSODE must be initialized each time it is called (ISTATE = 1) so that the order of the integration is therefore reset to 1 (the backward Euler method).

Subroutine DERV computes the first and second derivatives (UX and UXX) via a call to a cubic spline subroutine NCSPLE and then uses them to assemble the right-hand side of the PDE in the usual method of lines format. For example, the code segment

```
      NM1=N-1
      DO 1 I=2,NM1
      CT(I)=2.0D+00*VIS*CXX(I)-CA(I)*CX(I)
1     CONTINUE
```

was used to define the right-hand side of Burgers' equation

$$u_t = \epsilon u_{xx} - u u_x$$

in the interior of the space interval.

Subroutine ANUGB1 generates a nonuniform grid adaptively from a uniform grid using a second-derivative monitor function criterion. To generate the new nonuniform grid, grid points are added whenever the magnitude of the second derivative of the dependent variable exceeds a level defined by the user. The first call to ANUGB1 is preceded by either

(1) a call to subroutine GRIDE to generate a uniform grid as an input to ANUGB1, which is termed the basic grid, or
(2) a call to subroutine GRIDNA to generate a nonuniform grid as an input to ANUGB1.

Subsequent calls to ANUGB1 start with a nonuniform grid generated by the previous call.

The set of equidistributed grid points, termed *basic nodes*, which are created on the first call to ANUGB1 are retained throughout the entire solution process. When constructing adaptive grids, extra nodes are inserted between the basic nodes and these are referred to as *adaptive nodes*. The adaptive nodes may be added or removed at each regridding, but the basic nodes are always included. In general, the initial basic grid and succeeding adaptive grids will have different numbers of points, and of course, different grid spacings.

The user sets parameters (XFL and XFR) giving the maximum intervals of the independent variable to the left and right of a basic grid point into which one or more adaptive grid points may be inserted. Beyond these intervals, however, the value of the second derivative will have no effect on the insertion of adaptive grid points. The user also sets the number of intervals to be allowed in the adaptive grid between two basic grid points.

2.5. DSS/3—A Modification of DSS/2

The adaption process in DSS/2 is performed at predefined times only and is decoupled from the integration process. This type of adaption is known as static regridding. The sample program for DSS/2 in [8] suggests regridding approximately 50 times during the integration run and these regriddings are performed whether they are necessary or not. As mentioned earlier, the integrator has to be reinitialised after each regridding since the time step history is no longer useful to LSODE [9,10]. The newly inserted adaptive nodes take their initial values from the cubic spline interpolator.

DSS/2 was designed as a teaching tool where efficiency (in terms of cpu) was not of primary concern to the author. However, it is an excellent package and will be of great benefit to the scientific community and the author deserves our congratulations not just for his text on the subject, but also for his willingness to make available the codes described in the text to interested readers.

Our main motivation when modifying DSS/2 was to improve cpu times by minimizing the amount of work done to attain a given accuracy and also, if possible, to reduce the global error. To this end, we have made a number of subtle but effective changes to this code and we will refer to the modified code as DSS/3.

As a first step, we decided to change the way in which the semidiscretization is performed so as to allow banded Jacobians. We did this by using PDEONE [11] which employs finite difference quotients to approximate the spatial derivative terms. The finite difference approximations rely only on the neighbouring nodes, and hence, have a narrower "bandwidth" requirement than that for cubic splines. As a result of this modification, for the scalar problem for example, LSODE employed a tridiagonal Jacobian which dramatically reduced the number of function evaluations and cpu time.

A further improvement resulted by changing the way in which DSS/2 performed static regriddings. The original code adapted the grid at fixed time points resulting in a fixed number of regriddings, typically 50. Instead, we decided to test the solution after every time step and perform a regridding only if the current grid was "unsuitable." This proved very effective with the number of regriddings ranging from 0 to 15 for our test problems. An added bonus of fewer regriddings is the corresponding reduction in the number of times that LSODE has to be reinitialised.

2.6. NAG (Fixed Mesh) Routines

There are three routines available in the NAG Library for solving parabolic equations, D03PAF, D03PBF, D03PGF [12]. These are designed for equations in one space dimension and in time, and the equations may include nonlinear terms, provided the second-order space derivative u_{xx} and the time derivative u_t occur linearly. There are certain restrictions on the coefficients, intended to ensure that the equation is parabolic and not hyperbolic, and (in D03PAF, D03PBF) that integration takes place in the stable direction.

The method of solution is to discretize the space derivatives using finite differences on a fixed mesh, and to solve the resulting system of ordinary differential equations using Gear's method. The routine specifications are relatively lengthy, and the user is advised to try D03PAF first on a simple case, to see how the method works.

The three routines are as follows:

- D03PAF solves a **single** parabolic equation of the form

$$\frac{\partial u}{\partial t} = x^{-m}\frac{\partial}{\partial x}\left(x^m g(x,t,u)\frac{\partial u}{\partial x}\right) + f\left(x,t,u,\frac{\partial u}{\partial x}\right). \tag{21}$$

  The parameter m allows the routine to handle different coordinate systems easily (see specification). The boundary conditions on u must be given for $a < x < b$ at the initial time t, and for $x = a$, $x = b$ at subsequent times.
- D03PBF solves a system of parabolic equations of fairly simple form, with boundary conditions as above. It will also handle a mixture of elliptic and parabolic equations, in which the time derivative is absent from some of the equations.
- D03PGF is a more general routine, which solves a system of parabolic (and possibly elliptic) equations, with fewer restrictions than D03PBF. It also provides additional facilities for monitoring the solution and for output.

In the numerical experiments to follow, we compare our results with those obtained from D03PGF using a uniform (spatial) mesh. It is an efficient solver, so we can use its results as a benchmark for our adaptive schemes.

Table 1. Key to the statistics collected for each method.

NSTEP	The number of time steps used.
NFE	The number of function evaluations by the temporal integrator (excluding the cost of Jacobian evaluations).
NfCALLS	The number of calls to the subroutine defining the PDE.
NJE	The number of Jacobian evaluations.
GLOERR	The global error of the solution.
ENDERR	The local error of the solution on the last time step.
CPU	The total CPU time taken to solve the problem.
NRG	The number of static regriddings performed (DSS/2/3 only).
NF	The number of adaptive nodes to be added at any basic interval (DSS/2/3 only).
MAXNODES	The maximum number of nodes used to solve the problem (DSS/2/3 only).
ALPHA	The spatial smoothing parameters used to restrict the ratios of the finite difference grid spacing.
TAU	The temporal smoothing parameter.

3. NUMERICAL EXPERIMENTS

3.1. Set of Test Problems

P1. This is Example 2 in [13]:

$$u_t = u_{xx} + \pi^2 \sin(\pi x), \qquad 0 < x < 1, \quad 0 < t \leq 1,$$
$$u(0,t) = u(1,t) = 1, \qquad t \geq 0, \qquad u(x,0) = 1, \qquad 0 \leq x \leq 1,$$

whose exact solution is

$$u(x,t) = 1 + \left[1 - e^{-\pi^2 t}\right] \sin(\pi x).$$

P2. A linear heat conduction problem which appeared in Adjerid and Flaherty [14,15] is

$$u_t + u_x - u_{xx} = f(x,t), \qquad 0 \leq x \leq 1, \quad t > 0,$$

where $f(x,t)$, the initial function $u(x,0)$ and Dirichlet boundary conditions were chosen so that the exact solution is

$$u(x,t) = \frac{1}{2}\left[1 - \tanh\left\{C_1\left(x - C_2 t - C_3\right)\right\}\right].$$

The solution is a travelling wave and its steepness, speed, and phase can be determined by selecting C_1, C_2, and C_3. Following [16], we take $C_1 = 10$ and 100, $C_2 = 1$ and $C_3 = -0.15$.

P3. One of the best-known test problems is that of Burgers' equation (see [5,11,17–19])

$$u_t = \epsilon u_{xx} - u u_x, \qquad 0 < x < 1, \quad 0 < t \leq 1.$$

We consider the version of the problem which appears in [11], where the initial and Dirichlet boundary conditions are taken from the exact solution

$$u(x,t) = 1 - \frac{0.9 r_1}{R} - \frac{0.5 r_2}{R}, \qquad R = r_1 + r_2 + r_3,$$
$$r_1 = \exp\left(\frac{-x + 0.5 - 4.95t}{20\epsilon}\right), \qquad r_2 = \exp\left(\frac{-x + 0.5 - 0.75t}{4\epsilon}\right),$$
$$r_3 = \exp\left(\frac{-x + 0.375}{2\epsilon}\right),$$

Table 2. Results for Test Problem 1.

Integration using 81 nodes with atol = 10^{-3}						
METHOD	DSS/2	DSS/3	CWI	SFU	RPI	D03PGF
NSTEP	100	26	49	39	30	45
NFE	164	31	85	62	59	53
NFCALLS	428970	3634	15440	41396	179200	5265
NJE	65	5	12	14	13	4
GLOERR	2.0852e−3	1.7636e−3	3.9728e−3	1.4975e−3	1.3230e−4	4.0448e−3
ENDERR	1.6270e−4	1.2724e−4	3.5060e−4	8.4820e−5	1.3230e−4	1.2262e−4
CPU	41.88	0.71	4.88	10.91	51.32	1.05
NRG	50	0				
NF	2	2				
MAXNODES	81	81				
ALPHA			0.0100			
TAU			0.0010	0.3000		

Integration using 81 nodes with atol = 10^{-4}						
METHOD	DSS/2	DSS/3	CWI	SFU	RPI	D03PGF
NSTEP	175	38	85	56	41	69
NFE	263	47	149	94	75	84
NFCALLS	679953	5372	23440	38710	165760	8262
NJE	103	7	16	12	11	6
GLOERR	2.4410e−4	1.9102e−4	2.6262e−4	3.5207e−4	2.7129e−5	1.2043e−3
ENDERR	1.2658e−4	1.3989e−4	1.2292e−4	6.2130e−5	2.7129e−5	1.2472e−4
CPU	67.20	0.89	7.95	11.11	48.22	1.81
NRG	50	0				
NF	2	2				
MAXNODES	81	81				
ALPHA			0.0100			
TAU			0.0010	0.3000		

Integration using 81 nodes with atol = 10^{-5}						
METHOD	DSS/2	DSS/3	CWI	SFU	RPI	D03PGF
NSTEP	224	47	151	83	57	100
NFE	332	59	251	131	107	125
NFCALLS	755793	6557	37360	52061	250880	13041
NJE	114	8	24	16	17	12
GLOERR	1.2020e−4	1.2480e−4	1.7480e−4	8.0880e−5	2.7780e−6	1.2803e−4
ENDERR	1.1891e−4	1.2289e−4	1.2968e−4	8.0880e−5	2.7780e−6	1.2803e−4
CPU	74.18	1.18	12.79	15.00	72.25	2.69
NRG	50	0				
NF	2	2				
MAXNODES	81	81				
ALPHA			0.0100			
TAU			0.0010	0.3000		

for different values of ϵ. The solution exhibits a sharp moving wavefront with increasing time whose steepness depends on ϵ. As noted in [11], we have found that, on a uniform mesh, typically 200 to 400 points are required for an accurate simulation of the solution.

Table 3. Results for Test Problem 2.

Integration using 21 nodes with atol = 10^{-5} and $C_1 = 10.0$						
METHOD	DSS/2	DSS/3	CWI	SFU	RPI	D03PGF
NSTEP	842	255	190	187	107	194
NFE	1150	327	349	380	212	247
NFCALLS	178114	13656	12920	26030	49420	5376
NJE	268	52	33	30	7	3
GLOERR	2.0711e−2	9.3905e−3	7.6098e−3	1.4558e−2	2.2904e−5	1.4050e−2
ENDERR	1.5005e−2	1.8239e−3	7.5581e−3	1.2711e−2	2.2904e−5	7.8756e−3
CPU	22.75	3.28	5.98	11.81	26.43	1.45
NRG	50	5				
NF	2	2				
MAXNODES	27	34				
ALPHA			0.0100			
TAU			0.0010	0.0010		

Integration using 41 nodes with atol = 10^{-5} and $C_1 = 10.0$						
METHOD	DSS/2	DSS/3	CWI	SFU	RPI	D03PGF
NSTEP	755	234	242	228	91	186
NFE	1105	322	416	453	178	200
NFCALLS	658717	27610	28880	66573	89320	8692
NJE	269	57	34	38	7	4
GLOERR	2.7716e−3	2.4762e−3	4.1635e−3	4.4475e−3	1.4578e−5	3.3779e−3
ENDERR	2.1377e−3	5.0126e−4	4.0977e−3	3.7748e−3	1.4578e−5	2.0064e−3
CPU	89.11	5.49	12.61	25.38	47.29	2.20
NRG	50	6				
NF	2	2				
MAXNODES	52	65				
ALPHA			0.0100			
TAU			0.0010	0.0010		

Integration using 81 nodes with atol = 10^{-5} and $C_1 = 10.0$						
METHOD	DSS/2	DSS/3	CWI	SFU	RPI	D03PGF
NSTEP	604	203	272	248	118	191
NFE	889	291	515	488	233	207
NFCALLS	1936698	48832	70720	153260	209440	18225
NJE	205	46	41	44	7	6
GLOERR	6.3910e−4	6.5322e−4	2.1075e−3	1.2351e−3	1.5939e−4	8.3674e−4
ENDERR	3.0107e−4	1.5508e−4	2.0651e−3	1.0225e−3	1.5939e−4	4.9882e−4
CPU	335.50	8.93	28.76	52.81	110.65	4.19
NRG	50	7				
NF	2	2				
MAXNODES	100	128				
ALPHA			0.0100			
TAU			0.0010	0.0010		

P4. An equation which models a single-step reaction with diffusion (see [5,20] for example) is

$$u_t = u_{xx} + D(1+\alpha-u)e^{-\delta/u}, \qquad 0 < x < 1, \quad t > 0,$$
$$u_x(0,t) = 0, \qquad u(1,t) = 1, \qquad u(x,0) = 1, \qquad 0 \le x \le 1,$$

where $D = (R/\alpha\delta)e^{\delta}$. The problem was solved on a time interval from $t = 0$ to $t = 0.29$. Initially, the temperature starts out at 1 and gradually increases, with a maximum at $x = 0$. At a finite

Table 4. Results for Test Problem 2.

Integration using 21 nodes with atol = 10^{-5} and $C_1 = 100.0$					
METHOD	DSS/2	DSS/3	CWI	SFU	D03PGF
NSTEP	2247	1571	321	439	2463
NFE	2775	2286	1175	3242	3494
NFCALLS	353694	75756	23500	61598	73374
NJE	342	136	65	73	82
GLOERR	2.2012e+0	2.6893e+0	2.3851e−2	1.4929e−1	9.0983e+0
ENDERR	1.9787e+0	1.9756e+0	2.0397e−2	5.5891e−2	7.0780e+0
CPU	35.72	18.00	9.62	23.69	19.37
NRG	50	5			
MAXNODES	33	39			
NF	2	2			
TAU			0.001	0.001	
ALPHA			0.01		

Integration using 41 nodes with atol = 10^{-5} and $C_1 = 100.0$					
METHOD	DSS/2	DSS/3	CWI	SFU	D03PGF
NSTEP	2744	1617	293	565	2704
NFE	3483	2354	1105	4751	3801
NFCALLS	991139	126146	44200	185289	155841
NJE	445	150	59	108	89
GLOERR	2.2320e−1	2.2243e−1	9.6122e−3	4.8385e−2	2.6907e+0
ENDERR	1.9857e−1	1.0243e−1	7.7402e−3	3.1007e−2	1.9756e+0
CPU	100.97	27.10	16.17	60.13	36.57
NRG	50	6			
MAXNODES	47	60			
NF	2	2			
TAU			0.001	0.001	
ALPHA			0.01		

Integration using 81 nodes with atol = 10^{-5} and $C_1 = 100.0$					
METHOD	DSS/2	DSS/3	CWI	SFU	D03PGF
NSTEP	2658	1104	385	711	2743
NFE	3524	1635	1510	6718	3463
NFCALLS	4948242	170714	120720	530722	280503
NJE	629	103	84	158	14
GLOERR	3.3198e−1	2.5321e−2	5.4670e−3	3.3094e−2	2.2260e−1
ENDERR	5.7081e−2	1.3902e−2	4.3453e−3	1.4275e−2	1.0242e−1
CPU	638.86	31.95	42.84	153.80	63.87
NRG	50	6			
MAXNODES	90	112			
NF	2	2			
TAU			0.001	0.001	
ALPHA			0.01		

time, ignition occurs and the temperature at $x = 0$ increases rapidly to $1 + \alpha$. A steep front then forms and propagates towards $x = 1$ with speed proportional to $(1/2(1 + \alpha))e^{\alpha\delta}$. The problem reaches a steady state once the flame propagates to $x = 1$. Following [5], we solve the problem with $\alpha = 1$, $\delta = 20$ and $R = 5$.

Table 5. Results for Test Problem 3.

Integration using 41 nodes with atol = 10^{-5} and $\epsilon = 3 \times 10^{-3}$						
METHOD	DSS/2	DSS/3	CWI	SFU	RPI	D03PGF
NSTEP	572	228	249	228	334	291
NFE	774	258	462	419	796	355
NFCALLS	389483	19164	38640	67821	329560	15047
NJE	152	18	56	40	19	4
GLOERR	5.7140e−2	8.8595e−2	7.4050e−2	4.3411e−2	4.6117e−3	2.2234e−1
ENDERR	5.7140e−2	6.9151e−2	1.2228e−3	3.1634e−3	4.6117e−3	2.0255e−1
CPU	27.08	3.20	12.61	20.06	78.89	2.93
NRG	50	3				
NF	2	2				
MAXNODES	54	68				
ALPHA			0.0100			
TAU			0.0010	0.0010		

Integration using 81 nodes with atol = 10^{-5} and $\epsilon = 3 \times 10^{-3}$						
METHOD	DSS/2	DSS/3	CWI	SFU	RPI	D03PGF
NSTEP	592	260	250	262	353	426
NFE	792	300	477	494	797	456
NFCALLS	1508986	43666	74160	185018	626080	37908
NJE	151	19	50	56	16	4
GLOERR	4.7670e−3	2.4957e−2	2.8608e−2	1.3444e−2	7.2476e−4	8.8291e−2
ENDERR	4.7670e−3	2.3566e−2	5.7563e−4	1.9939e−3	7.2476e−4	6.6479e−2
CPU	162.68	6.53	22.88	47.66	150.52	7.17
NRG	50	3				
NF	2	2				
MAXNODES	108	136				
ALPHA			0.0100			
TAU			0.0010	0.0010		

Integration using 161 nodes with atol = 10^{-5} and $\epsilon = 3 \times 10^{-3}$						
METHOD	DSS/2	DSS/3	CWI	SFU	RPI	D03PGF
NSTEP	594	291	313	305	346	513
NFE	794	329	596	586	689	612
NFCALLS	5970113	95147	183200	392253	952000	100947
NJE	151	20	61	57	8	5
GLOERR	1.2716e−3	6.2959e−3	1.2879e−2	3.7960e−3	4.3619e−4	2.4855e−2
ENDERR	1.2436e−3	6.2332e−3	8.1251e−4	1.3444e−3	4.3619e−4	2.2470e−2
CPU	1362.76	13.24	55.00	100.58	237.25	17.51
NRG	50	3				
NF	2	2				
MAXNODES	216	270				
ALPHA			0.0100			
TAU			0.0010	0.0010		

3.2. Numerical Results

P1. This problem has been included to allow the reader to see how the methods cope with a request for increasing accuracy in the temporal integration. For this reason, the grid size is held fixed at 81 nodes and the absolute temporal tolerance is varied from 10^{-3} to 10^{-5}.

From Table 2, we see that DSS/3 is the fastest of the six methods, but as might be expected for a problem with a smooth solution, the uniform mesh solver D03PGF provides accurate solutions and is faster than the remaining five methods for all tolerance values. RPI is the most accurate solver but, because of the underlying computational expense, it is the least efficient in terms of cpu and NfCALLS. The disadvantage of an automatic regridding of 50 times per integration run is clearly evident in the results for DSS/2 since a comparison with those for DSS/3 reveals that regridding is unnecessary for a problem with a very smooth solution. Even without regridding, the automatic call to ANUGB1 necessitates a further Jacobian evaluation.

P2. We solved Problem 2 for a fixed absolute temporal tolerance of 10^{-5} over a range of mesh densities (21, 41, and 81) for two values of C_1, namely 10 and 100, and this problem proved to be a more difficult test for all methods.

(a) The results for $C_1 = 10$ are summarised in Table 3. Surprisingly, the NAG code, D03PGF, was by far the most efficient (in terms of NfCALLS and cpu) and, with the exception of RPI, the most accurate of the six methods. As with most other problems, the RPI code produced the most accurate solutions. Of the adaptive methods, DSS/3 was the fastest for all mesh densities and was slightly more accurate than the remaining four methods. However, both DSS codes are expensive for this problem in terms of Jacobian evaluations.

(b) The results for $C_1 = 100$ are summarised in Table 4 and contrast dramatically with those in Table 3. Both CWI and SFU cope best with this problem using both 21 and 41 spatial nodes, but for 81 nodes, DSS/3 is the most efficient in terms of cpu time. Note that CWI requires significantly fewer NFE and NJE counts than SFU for all mesh densities. For this problem, the (fixed spatial mesh) NAG solver proves least able to cope with the solution-resolution difficulties and is least accurate of all the methods for the three mesh densities selected.

P3. Burgers' equation is arguably the most popular test problem in the literature of the adaptive mesh solution of one-dimensional parabolic partial differential equations. In applying the different methods to approximate the solution of this problem, we conducted three different experiments.

(a) Keeping the absolute temporal tolerance fixed at 10^{-5}, we examined the relative performances of the six methods using a range of mesh densities (41, 81, and 161 points, respectively). We can see from Table 5 that most methods returned comparable statistics with DSS/3 being the most efficient at the different mesh densities, but less accurate than either RPI, DSS/2, or SFU. D03PGF is competitive with DSS/3 in terms of cpu, but is less accurate for all tolerances.

(b) Keeping the number of spatial points fixed (161), as well as the absolute temporal tolerance (10^{-5}), we examined the relative performances of the six methods for decreasing values of ϵ, namely 2×10^{-3}, 1×10^{-3}, and 5×10^{-4}. The results are presented in Table 6. RPI was unable to produce a solution for the latter two values of ϵ, and with the exception of DSS/2, was least competitive with all the other methods for $\epsilon = 2 \times 10^{-3}$. As before, DSS/3 and D03PGF were most competitive in terms of cpu while SFU was the most accurate of all methods.

(c) With $\epsilon = 10^{-4}$, and again keeping the absolute temporal tolerance fixed at 10^{-5}, we examined the relative performances of the six methods using a range of mesh densities (41, 81, and 161 points, respectively). The results in Table 7 show that both DSS/3 and NAG are unable to cope at the coarser mesh density (41 points) and NAG is completely inaccurate also for 81 mesh points and produces a wildly oscillating solution. For 161 nodes, SFU is the most accurate solver, but is less efficient than DSS/3 and D03PGF in terms of cpu.

P4. Following the discussion of Problem 4 in [5], we examined the accuracy of all methods at the point $x = 0.0$, $t = 0.26$ where we calculated the reference solution to be $u(0, 0.26) \approx 1.6165343$. As

Table 6. Results for Test Problem 3.

Integration using 161 nodes, with atol = 10^{-5} and $\epsilon = 2 \times 10^{-3}$						
METHOD	DSS/2	DSS/3	CWI	SFU	RPI	D03PGF
NSTEP	716	407	350	361	452	678
NFE	960	458	1442	723	1363	828
NFCALLS	5680936	130176	230720	492741	1526560	137172
NJE	151	28	86	72	16	8
GLOERR	2.6406e−3	1.4074e−2	2.3572e−2	9.9983e−3	7.1606e−4	5.3814e−2
ENDERR	2.6284e−3	1.3322e−2	5.7214e−4	1.3555e−3	7.1606e−4	4.1122e−2
CPU	2633.93	51.19	193.81	122.61	368.33	66.70
NRG	50	3				
NF	2	2				
MAXNODES	208	266				
ALPHA			0.0100	0.0100	0.0100	0.0100
TAU			0.0010	0.0010		

Integration using 161 nodes, with atol = 10^{-5} and $\epsilon = 1 \times 10^{-3}$					
METHOD	DSS/2	DSS/3	CWI	SFU	D03PGF
NSTEP	1038	662	466	531	1018
NFE	1328	741	1971	1032	1056
NFCALLS	5301296	203573	315360	709776	173397
NJE	151	40	114	104	7
GLOERR	8.5439e−3	5.3856e−2	5.8669e−2	3.2829e−2	1.5878e−1
ENDERR	8.4400e−3	4.9677e−2	2.1539e−3	1.0895e−3	1.3956e−1
CPU	2387.42	79.28	261.04	175.55	88.94
NRG	50	3			
NF	2	2			
MAXNODES	196	260			
ALPHA			0.0100	0.0100	
TAU			0.0010		

Integration using 161 nodes, with atol = 10^{-5} and $\epsilon = 5 \times 10^{-4}$					
METHOD	DSS/2	DSS/3	CWI	SFU	D03PGF
NSTEP	1509	942	658	701	1222
NFE	1928	1041	3104	1418	1377
NFCALLS	7573606	288215	496640	970536	224112
NJE	222	57	186	142	5
GLOERR	3.8264e−2	1.5905e−1	1.3856e−1	9.1035e−2	3.0763e−1
ENDERR	2.9507e−2	1.4034e−1	1.8853e−3	1.4848e−3	2.7496e−1
CPU	3427.57	112.92	392.82	239.74	114.35
NRG	50	3			
NF	2	2			
MAXNODES	195	260			
ALPHA			0.0100		
TAU			0.0010	0.0010	

an experiment, we solved the problem using 321 spatial points and varied the absolute temporal tolerance from 10^{-6} to 10^{-8}. The results are presented in Table 8 and note that we were unable to obtain a solution from DSS/2 for all tolerances, while RPI was unable to solve the problems at tolerances below 10^{-6}. Of the remaining methods, DSS/3 is the most competitive in terms of cpu even though CWI requires fewer NfCALLS. At the finer tolerance level, both DSS/3 and

Table 7. Results for Test Problem 3.

Integration using 41 nodes with atol = 10^{-5} and $\epsilon = 10^{-4}$					
METHOD	DSS/2	DSS/3	CWI	SFU	D03PGF
NSTEP	1134	448	2061	552	403
NFE	1458	598	10475	3799	546
NFCALLS	1114547	43192	419000	148161	22386
NJE	210	30	665	83	8
GLOERR	9.2954e−1	1.4438e+0	3.8727e−1	5.2141e−1	1.0776e+0
ENDERR	9.0501e−1	1.4355e+0	1.1724e−2	2.1394e−2	7.1370e−1
CPU	97.83	6.83	121.99	42.87	4.48
NRG	50	3			
MAXNODES	79	81			
NF	2	2			
TAU			0.001	0.001	
ALPHA			0.01		

Integration using 81 nodes with atol = 10^{-5} and $\epsilon = 10^{-4}$					
METHOD	DSS/2	DSS/3	CWI	SFU	D03PGF
NSTEP	1669	712	783	788	817
NFE	2027	931	3602	6320	982
NFCALLS	4069098	133385	288160	499280	79542
NJE	209	45	225	144	6
GLOERR	9.3234e−1	9.4322e−1	3.2797e−1	3.2112e−1	1.4436e+0
ENDERR	9.3040e−1	8.8518e−1	1.8026e−3	2.5776e−3	1.4358e+0
CPU	640.61	19.65	83.27	132.50	15.76
NRG	50	4			
MAXNODES	148	161			
NF	2	2			
TAU			0.001	0.001	
ALPHA			0.01		

Integration using 161 nodes with atol = 10^{-5} and $\epsilon = 10^{-4}$				
METHOD	DSS/2	DSS/3	SFU	D03PGF
NSTEP	2502	1305	1248	1494
NFE	64577	1676	12431	1609
NFCALLS	13227860	427338	1976529	259049
NJE	298	75	296	6
GLOERR	9.0604e−1	4.7774e−1	2.6447e−1	9.4867e−1
ENDERR	8.6977e−1	3.4810e−1	5.0931e−3	8.8618e−1
CPU	24265.33	167.53	477.58	132.36
NRG	50	3		
MAXNODES	217	261		
DL	20.0	20.0		
XFL	0.02	0.15		
XFR	0.03	0.15		
NF	2	2		
TAU			0.001	

D03PGF provide accurate approximations to the reference solution, but the latter is the more accurate method at all tolerance values.

Table 8. Results for Test Problem 4.

Integration using 321 nodes with atol = 10^{-6}					
METHOD	DSS/3	CWI	SFU	RPI	D03PGF
NSTEP	474	341	433	417	765
NFE	636	1049	837	1295	930
NFCALLS	368107	335680	782826	2900800	336087
NJE	77	43	49	20	39
GLOERR	6.1537e−3	3.6965e−2	2.0279e−2	3.0260e−3	4.9280e−4
$u^h(0, 0.26)$	1.6215e+0	1.5796e+0	1.5963e+0	1.6196e+0	1.6170e+0
CPU	53.01	173.85	241.79	819.65	64.01
NRG	7				
NF	2				
MAXNODES	483				
ALPHA		0.0100			
TAU		0.0010	0.0010		

Integration using 321 nodes with atol = 10^{-7}					
METHOD	DSS/3	CWI	SFU	RPI	D03PGF
NSTEP	702	553	628	0	1146
NFE	897	1389	1179	0	1358
NFCALLS	502592	444480	986667	0	468660
NJE	98	51	58	0	34
GLOERR	1.9240e−3	3.8391e−2	2.1496e−2	0.0000e+0	1.1793e−4
$u^h(0, 0.26)$	1.6173e+0	1.5781e+0	1.5950e+0	0.0000e+0	1.6167e+0
CPU	73.15	255.61	317.85	0.00	90.78
NRG	7				
NF	2				
MAXNODES	482				
ALPHA		0.0100			
TAU		0.0010	0.0010		

Integration using 321 nodes with atol = 10^{-8}					
METHOD	DSS/3	CWI	SFU	RPI	D03PGF
NSTEP	970	836	905	0	1768
NFE	1185	1893	1638	0	2037
NFCALLS	668940	605760	1238358	0	693360
NJE	129	59	68	0	41
GLOERR	1.2084e−3	3.8439e−2	2.1862e−2	0.0000e+0	2.8300e−5
$u^h(0, 0.26)$	1.6166e+0	1.5781e+0	1.5947e+0	0.0000e+0	1.6165e+0
CPU	99.43	369.97	415.60	0.00	135.70
NRG	7				
NF	2				
MAXNODES	482				
ALPHA		0.0100			
TAU		0.0010	0.0010		
Reference solution at $x = 0$, $t = 0.26$: 1.6165343					

4. CONCLUDING REMARKS

The reader should note that DSS/3 was obtained from DSS/2 following a number of relatively simple adjustments in the way the latter package implemented the spatial discretization and mesh adaption. Other improvements are also possible. For example, from the tables of results, it can be

Table 9. Results for Test Problem 4.

Integration using 41 nodes with atol = 10^{-6}					
METHOD	DSS/3	CWI	SFU	RPI	D03PGF
NSTEP	454	292	303	428	768
NFE	801	816	1963	1338	1167
NFCALLS	43333	32640	76557	374640	47847
NJE	68	35	41	21	38
GLOERR	4.6812e−3	1.7775e−1	1.2769e−1	3.1438e−3	2.0223e−3
CPU	7.84	10.95	24.93	99.09	11.65
NRG	6				
MAXNODES	62				
$u^h(0, 0.26)$	1.6212e+0	1.4388e+0	1.4888e+0	1.6197e+0	1.6145e+0
NF	2				
TAU		0.001	0.001		
ALPHA		0.01			

Integration using 81 nodes with atol = 10^{-6}					
METHOD	DSS/3	CWI	SFU	RPI	D03PGF
NSTEP	470	300	383	436	721
NFE	835	890	2224	1350	923
NFCALLS	90693	71200	175696	756000	74763
NJE	68	35	45	21	29
GLOERR	5.1588e−3	1.1380e−1	7.4493e−2	3.0072e−3	2.7980e−4
CPU	14.41	22.85	53.85	199.89	18.00
NRG	6				
MAXNODES	123				
$u^h(0, 0.26)$	1.6217e+0	1.5027e+0	1.5420e+0	1.6195e+0	1.6168e+0
NF	2				
TAU		0.001	0.001		
ALPHA		0.01			

Integration using 161 nodes with atol = 10^{-6}					
METHOD	DSS/3	CWI	SFU	RPI	D03PGF
NSTEP	475	373	395	416	740
NFE	879	1049	2373	1295	994
NFCALLS	186940	167840	377307	1450400	160034
NJE	76	39	49	20	35
GLOERR	6.0969e−3	6.7643e−2	4.0178e−2	3.0197e−3	1.2418e−3
CPU	27.62	53.17	112.93	386.07	37.58
NRG	7				
MAXNODES	243				
$u^h(0, 0.26)$	1.6226e+0	1.5489e+0	1.5764e+0	1.6196e+0	1.6178e+0
NF	2				
TAU		0.001	0.001		
ALPHA		0.01			

seen that the value of MAXNODES is much greater for DSS/3 than for DSS/2. It is apparently an advantage to allow the number of nodes to increase in this way, and it can be accomplished through simple parameter changes in the organization of the adaptive mesh routine ANUGB1 within DSS/2. In the case of Burgers' equation, however, we have experimented, without success, with parameter changes in order to increase the number of regriddings so that accuracy would

be improved with a not too much greater increase in computational expense (DSS/2 is more accurate than DSS/3 for this problem).

As a key to understanding the great differences in cpu and NfCALL measurements for the various methods in Tables 2–8, it is instructive to examine the respective computational costs. Taking Problem 3 in Table 5, for example, with $N = 81$ and $TOL = 10^{-5}$, note that SFU requires 185,018 NfCALLS compared with 74,160 for CWI even though they are comparable schemes. The differences can be explained by the very different costs of function and Jacobian evaluations and the bandwidth requirement for each method. For any given scalar (Dirichlet) problem with N spatial nodes, the value of NfCALLS for each of the codes may be explained using simple notation. Let F and J denote single function and Jacobian evaluations, respectively. Then formulae for calculating total NfCALLS for α function evaluations and β Jacobian evaluations are as in the following table.

Computational costs in terms of NfCALLS for a Dirichlet problem with N spatial nodes				
		Number of NfCALLS		
Package	Jacobian bandwidth	Per function evaluation (F)	Per Jacobian evaluation (J)	For a total of $\alpha F + \beta J$
CWI	9	$N-1$	$9F = 9(N-1)$	$(\alpha + 9\beta)(N-1)$
DSS/2	N	$N-2$	$NF = N(N-2)$	$(\alpha + N\beta + 1)(N-2)$
DSS/3	3	$N-2$	$3F = 3(N-2)$	$(\alpha + 3\beta)(N-2)$
D03PGF	3	N	$3F = 3N$	$(\alpha + 3\beta)N$
RPI	20	$7(N-1)$	$20F = 140(N-1)$	$7(\alpha + 20\beta + 1)(N-1)$
SFU	33	$N-2$	$33F = 33(N-2)$	$(\alpha + 33\beta)(N-2)$

It can be seen that, because of the relatively large Jacobian bandwidths of both RPI (33) and SFU (20), both packages are necessarily very expensive in terms of NfCALLs. RPI incurs an additional expense as a result of the high cost of a function call, $7(N-1)$ compared with $N-2$, $N-1$, or N for all the other methods.

For many problems, as can be seen from the numerical results, the NAG routine D03PGF is competitive with all the adaptive solvers (including DSS/3 for most problems) even though it is using a fixed, uniform spatial mesh. However, the results for Problems 2 and 3 (Tables 4 and 7, respectively) clearly illustrate the limitations of a fixed spatial mesh solver when applied to approximate rapidly varying solutions.

Overall, the modified scheme, DSS/3, has proved more accurate and more efficient in cpu usage than all the other adaptive methods over a limited but representative set of test problems. The relatively large bandwidths of the Jacobian matrices for CWI, SFU, and RPI (particularly in the latter case) has adversely affected execution times. However, as with DSS/2, these solvers are reliable, robust, and give accurate solutions to all problems.

REFERENCES

1. J.G. Verwer, J.G. Blom, R.M. Furzeland and P.A. Zegeling, A moving-grid method for one-dimensional PDEs based on the method of lines, Technical Report NM-R8818, Centre for Mathematics and Computer Science (CWI), Amsterdam, (1988).
2. L.R. Petzold, A description of DASSL: A differential/algebraic system solver, In *Scientific Computing*, (Edited by R.S. Stepleman), pp. 65–68, North Holland, (1983).
3. J.G. Blom and P.A. Zegeling, A moving-grid interface for systems of one-dimensional time-dependent partial differential equations, Technical Report NM-R8904, Centre for Mathematics and Computer Science (CWI), Amsterdam, (1989).
4. R.D. Skeel and M. Berzins, A method for the spatial discretization of parabolic equations in one space variable, *SIAM J. Sci. Stat. Comp.* **11**, 1–32 (1990).

5. W. Huang, Y. Ren and R.D. Russell, Moving mesh methods based on moving mesh partial differential equations, Technical Report 92-15, Simon Fraser University, Department of Mathematics, Burnaby, BC, Canada, (October 1992).
6. S. Adjerid and J.E. Flaherty, A moving mesh finite element method with local refinement for parabolic partial differential equations, *Comp. Meths. Appl. Mech. Engr.* **56**, 3–26 (1986).
7. K. Miller and R.N. Miller, Moving finite elements I, *SIAM J. Numer. Anal.* **18**, 1019–1032 (1981).
8. W.E. Schiesser, *The Numerical Method of Lines*, Academic Press, London, (1991).
9. A.C. Hindmarsh, LSODE and LSODI: Two new initial-value ordinary differential equation solvers, *ACM SIGNUM Newsletter* **15**, 10–11 (1980).
10. A.C. Hindmarsh, ODEPACK: A systematized collection of ODE solvers, In *Scientific Computing*, (Edited by R.S. Stepleman), pp. 55–64, IMACS, North-Holland, Amsterdam, (1983).
11. R.F. Sincovec and N.K. Madsen, Software for nonlinear partial differential equations, *ACM Trans. Math. Software* **1**, 232–260 (1975).
12. P.M. Dew and J.E. Walsh, A set of library routines for solving parabolic equations in one space dimension, *ACM Trans. Math. Software* **7**, 295–314 (1981).
13. N.K. Madsen and R.F. Sincovec, Algorithm 540: PDECOL general collocation software for partial differential equations, *ACM Trans. Math. Software* **5**, 326–351 (1979).
14. S. Adjerid and J.E. Flaherty, Adaptive finite element methods for parabolic systems in one and two dimensions, Technical Report 86-20, Rensselaer Polytechnic Institute, Troy, NY, (September 1986).
15. J.E. Flaherty, P.K. Moore and C. Ozturan, Adaptive overlapping grid methods for parabolic systems, In *Adaptive Methods for Partial Differential Equations*, (Edited by J. Paslow, M.S. Shepherd and J.D. Vasilakis), pp. 176–193, SIAM, Philadelphia, (1989).
16. S. Adjerid and J.E. Flaherty, A moving finite element method with error estimation and refinement for one-dimensional time dependent partial differential equations, *SIAM J. Numer. Anal.*, **23**, 778–796 (1986).
17. H. Aref and P.K. Darpia, Note on finite difference approximations to Burgers' equation, *SIAM J. Sci. Stat. Comp.* **5**, 856–863 (1984).
18. J.G. Blom, J.M. Sanz-Serna and J.G. Verwer, On simple moving grid methods for one-dimensional evolutionary partial differential equations, *J. Comput. Phys.* **74**, 191–213 (1988).
19. S.F. Davis and J.E. Flaherty, An adaptive finite element method for initial-boundary value problems for parabolic differential equations, *SIAM J. Sci. Stat. Comp.* **3**, 6–27 (1982).
20. L.R. Petzold, Observations on an adaptive moving grid method for one-dimensional systems of partial differential equations, *Appl. Numerical Maths.* **3**, 347–360 (1987).
21. E.A. Dorfi and L.O. Drury, Simple adaptive grids for 1-D initial value problems, *J. Comput. Phys.* **69**, 175–195 (1987).
22. W. Huang, Y. Ren and R.D. Russell, Moving mesh partial differential equations (MMPDEs) based on the equidistribution principle, (submitted for publication) (1992).